JDO XML MetaData Reference (v5.2)

Table of Contents

Metadata for package tag . . . 7
Metadata for class tag . . . 8
Metadata for datastore-identity tag . . . 9
Metadata for primary-key tag . . . 11
Metadata for inheritance tag . . . 12
Metadata for discriminator tag . . . 13
Metadata for version tag . . . 14
Metadata for query tag . . . 15
Metadata for field tag . . . 16
Metadata for property tag . . . 21
Metadata for fetch-group tag . . . 26
Metadata for embedded tag . . . 27
Metadata for key tag . . . 28
Metadata for value tag . . . 29
Metadata for order tag . . . 30
Metadata for index tag . . . 31
Metadata for foreign-key tag . . . 32
Metadata for unique tag . . . 33
Metadata for column tag . . . 34
Metadata for join tag . . . 36
Metadata for element tag . . . 37
Metadata for collection tag . . . 38
Metadata for map tag . . . 39
Metadata for array tag . . . 41
Metadata for sequence tag . . . 42
Metadata for fetch-plan tag . . . 44
Metadata for class extension tag . . . 45
Metadata for extension tag . . . 46

JDO has always accepted Metadata in XML format. As described in the the Mapping Guide this has to be contained in files with particular filenames in particular locations (relative to the name of the class), and that this metadata is *discovered* at runtime. You can provide JDO XML metadata, or alternatively ORM XML metadata, but with virtually identical format (the only difference being the top level element being *jdo* in the former case and *orm* in the latter case).

This page defines the format of the XML Metadata and can be used for either the JDO XML metadata or the ORM XML metadata.

Here is an example header for `package.jdo` files with JDO XSD specification

```
<?xml version="1.0" encoding="UTF-8" ?>
<jdo xmlns="http://xmlns.jcp.org/xml/ns/jdo/jdo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/jdo/jdo
http://xmlns.jcp.org/xml/ns/jdo/jdo_3_1.xsd" version="3.1">
    ...
</jdo>
```

Here is an example header for **package.orm** files with ORM XSD specification

```
<?xml version="1.0" encoding="UTF-8" ?>
<orm xmlns="http://xmlns.jcp.org/xml/ns/jdo/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/jdo/orm
http://xmlns.jcp.org/xml/ns/jdo/orm_3_0.xsd" version="3.0">
    ...
</orm>
```

JDO expects the XML metadata to be specified in a file or files in particular locations in the CLASSPATH. For example, if you have a class `com.mycompany.sample.MyExample`, JDO will look for any of the following resources until it finds one (in the order stated) :-

```
META-INF/package.jdo
WEB-INF/package.jdo
package.jdo
com/package.jdo
com/mycompany/package.jdo
com/mycompany/sample/package.jdo
com/mycompany/sample/MyExample.jdo
```

In addition to the above, you can split your metadata definitions between JDO XML MetaData files. For example if you have the following classes

```
com/mycompany/A.java
com/mycompany/B.java
com/mycompany/C.java
com/mycompany/app1/D.java
com/mycompany/app1/E.java
```

You could define the metadata for these 5 classes in many ways — for example put all class definitions in **com/mycompany/package.jdo**, or put the definitions for D and E in **com/mycompany/app1/package.jdo** and the definitions for A, B, C in **com/mycompany/package.jdo**, or have some in their class named MetaData files e.g **com/mycompany/app1/A.jdo**, or a mixture of the above. DataNucleus will always search for the metadata file containing the class definition for the class that it requires.

As well as JDO XML metadata, you can (also) use ORM XML metadata to override particular datastore-specific things like table and column names. JDO expects any ORM XML metadata to be specified in a file or files in particular locations in the CLASSPATH. These filenames depend on the **javax.jdo.option.mapping** persistence property. For example, if you have a class `com.mycompany.sample.MyExample`, and the persistence property is set to "mysql" then JDO will look for any of the following resources until it finds one (in the order stated) :-

```
META-INF/package-mysql.orm
WEB-INF/package-mysql.orm
package-mysql.orm
com/package-mysql.orm
com/mycompany/package-mysql.orm
com/mycompany/sample/package-mysql.orm
com/mycompany/sample/MyExample-mysql.orm
```

If your application doesn't make use of ORM metadata then you could turn off the searches for ORM Metadata files when a class is loaded up. You do this with the persistence property **datanucleus.metadata.supportORM** setting it to false.

By default any XML Metadata (JDO or ORM) will be validated for accuracy when loading it. Obviously XML is defined by a DTD or XSD schema and so should follow that. You can turn off such validations by setting the persistence property **datanucleus.metadata.xml.validate** to false when creating your PMF. Note that this only turns off the XML strictness validation, and *not* the checks on inconsistency of specification of relations etc.

By default DataNucleus will look for XML metadata in the above locations, as per the JDO spec. If

you set the persistence property **datanucleus.metadata.xml.allowJDO1_0** then the above list of locations will be extended to also include some locations that were in the original JDO 1.0.0 spec

```
META-INF/package.jdo
WEB-INF/package.jdo
package.jdo
com.jdo
com/package.jdo
com/mycompany.jdo
com/mycompany/package.jdo
com/mycompany/sample.jdo
com/mycompany/sample/package.jdo
com/mycompany/sample/MyExample.jdo
```

What follows provides a reference guide to XML MetaData elements for what goes in an XML metadata file. Refer to the relevant XSD for precise details.

- jdo
  - package
    - class
      - datastore-identity
        - column
        - extension
      - primary-key
        - column
      - inheritance
        - discriminator
          - column
        - join
          - column
      - version
        - column
        - extension
      - join
        - column
      - foreign-key
        - column
        - field
        - property

- index
  - column
  - field
  - property
- unique
  - column
  - field
  - property
- field
  - collection
    - extension
  - map
    - extension
  - array
  - join
    - primary-key
    - index
    - column
  - embedded
    - field
      - column
  - element
    - column
  - key
    - column
  - value
    - column
  - order
    - column
    - extension
  - column
    - extension
  - foreign-key
    - column
  - index

        - column
    - unique
        - column
    - extension
- property
    - collection
        - extension
    - map
        - extension
    - array
    - join
        - primary-key
        - index
        - column
    - embedded
        - field
            - column
    - element
        - column
    - key
        - column
    - value
        - column
    - order
        - column
    - column
        - extension
    - foreign-key
        - column
    - index
        - column
    - unique
        - column
    - extension
- fetch-group

        - field
      - query
    - sequence
      - extension
    - fetch-plan
    - extension
  - extension

Metadata for package tag

These are attributes within the <**package**> tag (jdo/package). This is used to denote a package, and all of the <**class**> elements that follow are in this Java package.

Attribute	Description	Values
name	Name of the java package	
catalog	Name of the catalog in which to persist the classes in this package. See also the property **javax.jdo.mapping.Catalog** in the PMF Guide.	
schema	Name of the schema in which to persist the classes in this package. See also the property **javax.jdo.mapping.Schema** in the PMF Guide.	

Metadata for class tag

These are attributes within the <**class**> tag (jdo/package/class). This is used to define the persistence definition for this class.

Attribute	Description	Values
name	Name of the class to persist	
identity-type	The identity type, specifying whether they are uniquely provided by the JDO implementation (datastore identity), accessible fields in the object (application identity), or not at all (nondurable identity).	**datastore**, application, nondurable
objectid-class	The class name of the primary key. When using **application identity**.	
requires-extent	Whether the JDO implementation must provide an Extent for this class.	**true**, false
detachable	Whether the class is detachable from the persistence graph.	true, **false**
embedded-only	Whether this class should only be stored embedded in the tables for other classes.	true, **false**
persistence-modifier	What type of persistability type this class exhibits. Please refer to JDO Class Types.	**persistence-capable**, persistence-aware, non-persistent
catalog	Name of the catalog in which to persist the class. See also the property **javax.jdo.mapping.Catalog** in the PMF Guide.	
schema	Name of the schema in which to persist the class. See also the property **javax.jdo.mapping.Schema** in the PMF Guide.	
table	Name of the table/view in which to persist the class. See also the property **datanucleus.identifier.case** in the Persistence Properties Guide.	
cacheable	Whether the class can be cached in a Level 2 cache.	**true**, false
serializeRead	Whether to default to locking objects of this type when reading them.	true, **false**

Metadata for datastore-identity tag

These are attributes within the <**datastore-identity**> tag (jdo/package/class/datastore-identity). This is used when the <**class**> to which this pertains uses datastore identity. It is used to define the precise definition of datastore identity to be used. This element can contain **column** sub-elements allowing definition of the column details where required - these are optional.

Attribute	Description	Values
strategy	Strategy for datastore identity generation for this class. *native* allows DataNucleus to choose the most suitable for the datastore. *sequence* will use a sequence (specified by the attribute **sequence**) - if supported by the datastore. *increment* will use the id values in the datastore to decide the next id. *uuid-string* will use a UUID string generator (16-characters). *uuid-hex* will use a UUID string generator (32-characters). *identity* will use a datastore inbuilt auto-incrementing types. *auid* is a DataNucleus extension, that is an almost universal id generator (best possible derivate of a DCE UUID). *max* is a DataNucleus extension, that uses "select max(column)+1 from table" for the identity. *timestamp* is a DataNucleus extension, providing the current timestamp. *timestamp-value* is a DataNucleus extension, providing the current timestamp millisecs. *[other values]* to utilise user-supplied DataNucleus value generator plugins.	**native**, sequence, increment, identity, uuid-string, uuid-hex, *auid*, *max*, *timestamp*, *timestamp-value*, _ [other values]_
sequence	Name of the sequence to use to generate identity values, when using a **strategy** of *sequence*. Please see also the **class** extension tags for controlling the sequence.	
column	Name of the column used for the datastore identity for this class.	

These are attributes within the <**extension**> tag (jdo/package/class/datastore-identity/extension). These are for controlling the generation of ids when in **datastore identity** mode.

Attribute	Description	Values
sequence-table-basis	This defines the basis on which to generate unique identities when using the TableValueGenerator (used by the "increment" strategy, and sometimes by "native"). You can either define identities unique against the base table name, or against the base class name (in an inheritance tree). Used when the strategy is set to *native* or *increment*	**class**, table
sequence-catalog-name	The catalog used to store sequences for use by value generators. See Value Generation. Default catalog for the datastore will be used if not specified.	

Attribute	Description	Values
sequence-schema-name	The schema used to store sequences for use by value generators. See Value Generation. Default schema for the datastore will be used if not specified.	
sequence-table-name	The table used to store sequences for use by value generators. See Value Generation.	**SEQUENCE_TABLE**
sequence-name-column-name	The column name in the sequence-table used to store the name of the sequence for use by value generators. See Value Generation.	**SEQUENCE_NAME**
sequence-nextval-column-name	The column name in the sequence-table used to store the next value in the sequence for use by value generators. See Value Generation.	**NEXT_VAL**
key-min-value	The minimum key value for use by value generators. Keys lower than this will not be generated. See Value Generation.	
key-max-value	The maximum key value for use by value generators. Keys higher than this will not be generated. See Value Generation.	
key-initial-value	The starting value for use by value generators. Keys will start from this value when being generated. See Value Generation.	
key-cache-size	The cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	
key-database-cache-size	The database cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	

Metadata for primary-key tag

These are attributes within the <**primary-key**> tag (jdo/package/class/primary-key or class/field/join/primary-key). It is used to specify the name of the primary key constraint in the datastore during the schema generation process. When used under <join> it specifies that the join table has a primary-key.

Attribute	Description	Values
name	Name of the primary key constraint.	
column	Name of the column to use for the primary key	

Metadata for inheritance tag

These are attributes within the <**inheritance**> tag (jdo/package/class/inheritance). It is used when this class is part of an inheritance tree, and to denote how the class is stored in the datastore since there are several ways (strategies) in which it can be stored.

Attribute	Description	Values
strategy	Strategy for inheritance of this class. Please refer to the Inheritance Guide.	new-table, subclass-table, superclass-table, complete-table

Metadata for discriminator tag

These are attributes within the <**discriminator**> tag (jdo/package/class/inheritance/discriminator). This is used to define a discriminator column that is used when this class is stored in the same table as another class in the same inheritance tree. The discriminator column will contain a value for objects of this class, and different values for objects of other classes in the inheritance tree.

Attribute	Description	Values
strategy	Strategy for the discrimination column	value-map, class-name, none
value	Value for the discrimination column	
column	Name for the discrimination column	
indexed	Whether the discriminator column should be indexed. This is to be specified when defining index information	true, false, unique

Metadata for version tag

These are attributes within the <**version**> tag (jdo/package/class/version). This is used to define whether and how this class is handled with respect to optimistic transactions.

Attribute	Description	Values
strategy	Strategy for versioning of this class. The "version-number" mode uses an incremental numbered value, and the "date-time" mode uses a java.sql.Timestamp value. *state-image* isn't currently supported.	state-image, date-time, version-number
column	Name of the column in the datastore to store this field	
indexed	Whether the version column should be indexed. This is to be specified when defining index information	true, false, unique

These are attributes within the <**extension**> tag (jdo/package/class/version/extension).

Attribute	Description	Values
field-name	This extension allows you to define a field that will be used to contain the version of the object. It is populated by DataNucleus at persist. See JDO Versioning	

Metadata for query tag

These are attributes within the <**query**> tag (jdo/package/class/query). This element is used to define any "named queries" that are to be available for this class. This element contains the query single-string form as its content.

Attribute	Description	Values
name	Name of the query. This name is mandatory and is used in calls to `pm.newNamedQuery()`. Has to be unique for this class.	
language	Query language to use. Some datastores offer other languages	**JDOQL**, SQL, JPQL
unique	Whether the query is to return a unique result (only for SQL queries).	true, **false**
result-class	Class name of any result class (only for SQL queries).	

Metadata for field tag

These are attributes within the <**field**> tag (jdo/package/class/field). This is used to define the persistence behaviour of the fields of the class to which it pertains. Certain types of fields are, by default, persisted. This element can be used to change the default behaviour and maybe not persist a field, or to persist something that normally isn't persisted. It is used, in addition, to define more details about how the field is persisted in the datastore.

Attribute	Description	Values
name	Name of the field.	
persistence-modifier	The persistence-modifier specifies how JDO manage each field in your persistent class. There are three options: persistent, transactional and none. **persistent** means that your field will managed by JDO and stored in the database on transaction commit. **transactional** means that your field will managed by JDO but not stored in the database; transactional fields values will be saved by JDO when you start your transaction and restored when you roll back your transaction. **none** means that your field will not be managed by JDO.	persistent, transactional, none
primary-key	Whether the field is part of any primary key (if using **application identity**).	true, **false**
null-value	How to treat null values of persistent fields during storage. Valid options are "exception", "default", "none" (where "none" is the default).	exception, default, **none**
default-fetch-group	Whether this field is part of the default fetch group for the class. Defaults to true for non-key fields of primitive types, java.util.Date, java.lang.**, java.math.**, etc.	**true**, false
embedded	Whether this field should be stored, if possible, as part of the object instead as its own object in the datastore. This defaults to true for primitive types, java.util.Date, java.lang.**, java.math.** etc and false for persistable, reference (Object, Interface) and container types.	true, false
serialized	Whether this field should be stored serialised into a single column of the table of the containing object.	true, **false**
dependent	Whether the field should be used to check for dependent objects, and to delete them when this object is deleted. In other words **cascade delete** capable.	true, **false**
mapped-by	The name of the field at the other end of a relationship. Used by 1-1, 1-N, M-N to mark a relation as bidirectional.	

Attribute	Description	Values
value-strategy	The strategy for populating values to this field. Is typically used for generating primary key values. See the definitions under "datastore-identity".	native, sequence, increment, identity, uuid-string, uuid-hex, *auid*, *max*, *timestamp*, *timestamp-value*, *[other values]*
sequence	Name of the sequence to use to generate values, when using a **strategy** of *sequence*. Please see also the **class** extension tags for controlling the sequence.	
recursion-depth	The depth that will be recursed when this field is self-referencing. Should be used alongside FetchPlan.setMaxFetchDepth() to control the objects fetched.	-1, **1**, 2, ... (integer)
field-type	Used to specify a more restrictive type than the field definition in the class. This might be required in order to map the field to the datastore. To be portable, specify the name of a single type that is itself able to be mapped to the datastore (e.g. a field of type Object can specify field-type="Integer").	
indexed	Whether the column(s) for this field should be indexed. This is to be specified when defining index information	true, false, unique
table	Table name to use for any join table overriding the default name provided by DataNucleus. This is used either for 1-N relationships with a join table or for Secondary Tables. See also the property **datanucleus.identifier.case** in the PMF Properties Guide.	
column	Column name to use for this field (alternative to specifying column sub-elements if only one column).	
delete-action	The foreign-key delete action. This is a shortcut to specifying foreign key information. Please refer to the <foreign-key> element for full details.	cascade, restrict, null, default, none
cacheable	Whether the field/property can be cached in a Level 2 cache.	**true**, false
load-fetch-group	Name of a fetch group to activate when a load of this field is initiated (due to it being currently unloaded). Not used for getObjectById, queries, extents etc. Better to use "fetch-group" and define your groups	
converter	Class name of a converter class (AttributeConverter) to use for this field.	

Attribute	Description	Values
use-default-conversion	Whether we should just use any default conversion (defined via persistent properties)	true, **false**

These are attributes within the <**extension**> tag (jdo/package/class/field/extension).

Attribute	Description	Values
cascade-persist	JDO defines that when an object is persisted then all fields will also be persisted using "persistence-by-reachability". This extension allows you to turn off the persistence of a field relation.	**true**, false
cascade-update	JDO defines that when an object is updated then all fields containing persistable objects will also be updated using "persistence-by-reachability". This extension allows you to turn off the update of a field relation.	**true**, false
cascade-refresh	When calling PersistenceManager.refresh() only fetch plan fields of the passed object will be refreshed. Setting this to true will refresh the fields of related PC objects in this field	true, **false**
allow-nulls	When the field is a collection by default it will not be allowed to have nulls present but you can allow them by setting this DataNucleus extension tag	true, **false**
insertable	Whether this field should be supplied when inserting into the datastore.	**true**, false
updateable	Whether this field should be supplied when updating the datastore.	**true**, false
implementation-classes	Used to define the possible classes implementing this interface/Object field. This is used to limit the possible tables that this is a foreign key to (when this field is specified as an interface/Object in the class). Value should be comma-separated list of fully-qualified class names	
key-implementation-classes	Used to define the possible classes implementing this interface/Object key. This is used to limit the possible tables that this is a foreign key to (when this key is specified as an interface/Object). Value should be comma-separated list of fully-qualified class names	
value-implementation-classes	Used to define the possible classes implementing this interface/Object value. This is used to limit the possible tables that this is a foreign key to (when this value is specified as an interface/Object). Value should be comma-separated list of fully-qualified class names	

Attribute	Description	Values
strategy-when-notnull	This is to be used in conjunction with the "value-strategy" attribute. Default JDO behaviour when you have a "value-strategy" defined for a field is to always create a strategy value for that field regardless of whether you have set the value of the field yourself. This extension allows you to only apply the strategy if the field is null at persistence. This extension has no effect on primitive field types (which can't be null) and the value-strategy will always be applied to such fields.	**true**, false
relation-discriminator-column	Name of a column to use for discrimination of the relation used by objects stored. This is defined when, for example, a join table is shared by multiple relations and the objects placed in the join table need discriminating for which relation they are for	**RELATION_DISCRIM**
relation-discriminator-pk	Whether the column added for the discrimination of relations is to be part of the PK when using a join table.	true, **false**
relation-discriminator-value	Value to use in the relation discriminator column for objects of this fields relation. This is defined when, for example, a join table is shared by multiple relations and the objects placed in the join table need discriminating for which relation they are for.	Fully-qualified class name
select-function	Permits to use a function when fetching contents from the database. A ? (question mark) is mandatory to have and will be replaced by the column name when generating the SQL statement. For example to specify a value of *UPPER(?)* will convert the field value to upper case on a datastore that supports that UPPER function. See Mapping : Column Adapters.	
insert-function	Permits to use a function when inserting into the database. A ? (question mark) is optional and will be replaced by the column name when generating the SQL statement. For example to specify a value of *TRIM(?)* will trim the field value on a datastore that supports that TRIM function. See Mapping : Column Adapters.	
update-function	Permits to use a function when updating into the database. A ? (question mark) is optional and will be replaced by the column name when generating the SQL statement. For example to specify a value of *FUNC(?)* will perform "FUNC" on the field value on a datastore that supports that FUNC function. See Mapping : Column Adapters.	

Attribute	Description	Values
sequence-table-basis	This defines the basis on which to generate unique identities when using the TableValueGenerator (used by the "increment" strategy, and sometimes by "native"). You can either define identities unique against the base table name, or against the base class name (in an inheritance tree). Used when the strategy is set to *native* or *increment*	**class**, table
sequence-catalog-name	The catalog used to store sequences for use by value generators. See Value Generation. Default catalog for the datastore will be used if not specified.	
sequence-schema-name	The schema used to store sequences for use by value generators. See Value Generation. Default schema for the datastore will be used if not specified.	
sequence-table-name	The table used to store sequences for use by value generators. See Value Generation.	**SEQUENCE_TABLE**
sequence-name-column-name	The column name in the sequence-table used to store the name of the sequence for use by value generators. See Value Generation.	**SEQUENCE_NAME**
sequence-nextval-column-name	The column name in the sequence-table used to store the next value in the sequence for use by value generators. See Value Generation.	**NEXT_VAL**
key-min-value	The minimum key value for use by value generators. Keys lower than this will not be generated. See Value Generation.	
key-max-value	The maximum key value for use by value generators. Keys higher than this will not be generated. See Value Generation.	
key-initial-value	The starting value for use by value generators. Keys will start from this value when being generated. See Value Generation.	
key-cache-size	The cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	
key-database-cache-size	The database cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	
mapping-class	Specifies the (java) mapping class to be used for mapping this field. This is only used where the user wants to override the default DataNucleus mapping class and provide their own mapping class for this field.	Fully-qualified class name

Metadata for property tag

These are attributes within the <**property**> tag (jdo/package/class/property). This is used to define the persistence behaviour of the Java Bean properties of the class to which it pertains. This element can be used to change the default behaviour and maybe not persist a property, or to persist something that normally isn't persisted. It is used, in addition, to define more details about how the property is persisted in the datastore.

Attribute	Description	Values
name	Name of the property. The "name" of a property is obtained by taking the getXXX, setXXX method names and using the XXX and making the first letter lowercase.	
persistence-modifier	The persistence-modifier specifies how to manage each property in your persistent class. There are three options: persistent, transactional and none. **persistent** means that your field will be managed and stored in the database on transaction commit. **transactional** means that your field will be managed but not stored in the database; transactional fields values will be saved by JDO when you start your transaction and restored when you roll back your transaction. **none** means that your field will not be managed.	persistent, transactional, none
primary-key	Whether the property is part of any primary key (if using **application identity**).	true, **false**
null-value	How to treat null values of persistent properties during storage.	exception, default, **none**
default-fetch-group	Whether this property is part of the default fetch group for the class. Defaults to true for non-key fields of primitive types, java.util.Date, java.lang.**, java.math.**, etc.	**true**, false
embedded	Whether this property should be stored, if possible, as part of the object instead as its own object in the datastore. This defaults to true for primitive types, java.util.Date, java.lang.**, java.math.** etc and false for persistable, reference (Object, Interface) and container types.	true, false
serialized	Whether this property should be stored serialised into a single column of the table of the containing object.	true, **false**
dependent	Whether the property should be used to check for dependent objects, and to delete them when this object is deleted. In other words **cascade delete** capable.	true, **false**
mapped-by	The name of the property at the other end of a relationship. Used by 1-1, 1-N, M-N to mark a relation as bidirectional.	

Attribute	Description	Values
value-strategy	The strategy for populating values to this property. Is typically used for generating primary key values. See the definitions under "datastore-identity".	native, sequence, increment, identity, uuid-string, uuid-hex, *auid*, *max*, *timestamp*, *timestamp-value*, *[other values]*
sequence	Name of the sequence to use to generate values, when using a **strategy** of *sequence*. Please see also the **class** extension tags for controlling the sequence.	
recursion-depth	The depth that will be recursed when this property is self-referencing. Should be used alongside `FetchPlan.setMaxFetchDepth()` to control the objects fetched.	-1, **1**, 2, ... (integer)
field-type	Used to specify a more restrictive type than the property definition in the class. This might be required in order to map the field to the datastore. To be portable, specify the name of a single type that is itself able to be mapped to the datastore (e.g. a field of type Object can specify field-type="Integer").	
indexed	Whether the column(s) for this property should be indexed. This is to be specified when defining index information	true, false, unique
table	Table name to use for any join table overriding the default name provided by DataNucleus. This is used either for 1-N relationships with a join table or for Secondary Tables. See also the property **datanucleus.identifier.case** in the PMF Properties Guide.	
column	Column name to use for this property (alternative to specifying column sub-elements if only one column).	
delete-action	The foreign-key delete action. This is a shortcut to specifying foreign key information. Please refer to the <foreign-key> element for full details.	cascade, restrict, null, default, none
cacheable	Whether the field/property can be cached in a Level 2 cache.	**true**, false
load-fetch-group	Name of a fetch group to activate when a load of this field is initiated (due to it being currently unloaded). Not used for getObjectById, queries, extents etc. Better to use "fetch-group" and define your groups	

These are attributes within the <**extension**> tag (jdo/package/class/property/extension).

Attribute	Description	Values
cascade-persist	JDO defines that when an object is persisted then all fields will also be persisted using "persistence-by-reachability". This extension allows you to turn off the persistence of a field relation.	**true**, false
cascade-update	JDO defines that when an object is updated then all fields containing persistable objects will also be updated using "persistence-by-reachability". This extension allows you to turn off the update of a field relation.	**true**, false
cascade-refresh	When calling PersistenceManager.refresh() only fetch plan fields of the passed object will be refreshed. Setting this to true will refresh the fields of related PC objects in this field	true, **false**
allow-nulls	When the field is a collection by default it will not be allowed to have nulls present but you can allow them by setting this DataNucleus extension tag	true, **false**
insertable	Whether this field should be supplied when inserting into the datastore.	**true**, false
updateable	Whether this field should be supplied when updating the datastore.	**true**, false
implementation-classes	Used to define the possible classes implementing this interface/Object field. This is used to limit the possible tables that this is a foreign key to (when this field is specified as an interface/Object in the class). Value should be comma-separated list of fully-qualified class names	
key-implementation-classes	Used to define the possible classes implementing this interface/Object key. This is used to limit the possible tables that this is a foreign key to (when this key is specified as an interface/Object). Value should be comma-separated list of fully-qualified class names	
value-implementation-classes	Used to define the possible classes implementing this interface/Object value. This is used to limit the possible tables that this is a foreign key to (when this value is specified as an interface/Object). Value should be comma-separated list of fully-qualified class names	
strategy-when-notnull	This is to be used in conjunction with the "value-strategy" attribute. Default JDO2 behaviour when you have a "value-strategy" defined for a field is to always create a strategy value for that field regardless of whether you have set the value of the field yourself. This extension allows you to only apply the strategy if the field is null at persistence. This extension has no effect on primitive field types (which can't be null) and the value-strategy will always be applied to such fields.	**true**, false

Attribute	Description	Values
relation-discriminator-column	Name of a column to use for discrimination of the relation used by objects stored. This is defined when, for example, a join table is shared by multiple relations and the objects placed in the join table need discriminating for which relation they are for	**RELATION_DISCRIM**
relation-discriminator-pk	Whether the column added for the discrimination of relations is to be part of the PK when using a join table.	true, **false**
relation-discriminator-value	Value to use in the relation discriminator column for objects of this fields relation. This is defined when, for example, a join table is shared by multiple relations and the objects placed in the join table need discriminating for which relation they are for.	Fully-qualified class name
select-function	Permits to use a function when fetching contents from the database. A ? (question mark) is mandatory to have and will be replaced by the column name when generating the SQL statement. For example to specify a value of *UPPER(?)* will convert to upper case the field value on a datastore that supports that UPPER function. See Mapping : Column Adapters.	
insert-function	Permits to use a function when inserting into the database. A ? (question mark) is optional and will be replaced by the column name when generating the SQL statement. For example to specify a value of *TRIM(?)* will trim the field value on a datastore that supports that TRIM function. See Mapping : Column Adapters.	
update-function	Permits to use a function when updating into the database. A ? (question mark) is optional and will be replaced by the column name when generating the SQL statement. For example to specify a value of *FUNC(?)* will perform FUNC() on the field value on a datastore that supports that FUNC function. See Mapping : Column Adapters.	
sequence-table-basis	This defines the basis on which to generate unique identities when using the TableValueGenerator (used by the "increment" strategy, and sometimes by "native"). You can either define identities unique against the base table name, or against the base class name (in an inheritance tree). Used when the strategy is set to *native* or *increment*	**class**, table
sequence-catalog-name	The catalog used to store sequences for use by value generators. See Value Generation. Default catalog for the datastore will be used if not specified.	
sequence-schema-name	The schema used to store sequences for use by value generators. See Value Generation. Default schema for the datastore will be used if not specified.	

Attribute	Description	Values
sequence-table-name	The table used to store sequences for use by value generators. See Value Generation.	**SEQUENCE_TABLE**
sequence-name-column-name	The column name in the sequence-table used to store the name of the sequence for use by value generators. See Value Generation.	**SEQUENCE_NAME**
sequence-nextval-column-name	The column name in the sequence-table used to store the next value in the sequence for use by value generators. See Value Generation.	**NEXT_VAL**
key-min-value	The minimum key value for use by value generators. Keys lower than this will not be generated. See Value Generation.	
key-max-value	The maximum key value for use by value generators. Keys higher than this will not be generated. See Value Generation.	
key-initial-value	The starting value for use by value generators. Keys will start from this value when being generated. See Value Generation.	
key-cache-size	The cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	
key-database-cache-size	The database cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	
mapping-class	Specifies the (java) mapping class to be used for mapping this field. This is only used where the user wants to override the default DataNucleus mapping class and provide their own mapping class for this field.	Fully-qualified class name

Metadata for fetch-group tag

These are attributes within the <**fetch-group**> tag (jdo/package/class/fetch-group). This element is used to define fetch groups that are utilised at runtime, and are of particular use with attach/detach. This element can contain **fetch-group** sub-elements allowing definition of hierarchical groups. It can also contain **field** elements, defining the fields that are part of this fetch-group.

Attribute	Description	Values
name	Name of the fetch group. Used with the fetch plan of the PersistenceManager.	
post-load	Whether to call jdoPostLoad when the fetch group is invoked.	true, false

Metadata for embedded tag

These are attributes within the <**embedded**> tag (jdo/package/class/embedded). It is used when this field is a persistable and is embedded into the same table as the class.

Attribute	Description	Values
owner-field	Name of the field in the embedded persistable that is the link back to the owning object (if any).	
null-indicator-column	Name of the column to be used for detacting if the embedded object is null.	
null-indicator-value	Value of the null-indicator-column that signifies that the embedded object is null.	

Metadata for key tag

These are attributes within the <**key**> tag (jdo/package/class/field/key). This element is used to define details for the persistence of a Map.

Attribute	Description	Values
mapped-by	When the map is formed by a foreign-key, the key can be a field in a value persistable class. This attribute defines which field in the value class is used as the key	
column	Name of the column (if only one)	
delete-action	Action to be performed when the owner object is deleted. This is to be specified when defining foreign key information	cascade, restrict, null, default, none
indexed	Whether the key column should be indexed. This is to be specified when defining index information	true, false, unique
unique	Whether the key column should be unique. This is to be specified when defining unique key information	true, false
converter	Class name of a converter class (AttributeConverter) to use for this key.	
use-default-conversion	Whether we should just use any default conversion (defined via persistent properties)	true, **false**

Metadata for value tag

These are attributes within the <**value**> tag (jdo/package/class/field/value). This element is used to define details for the persistence of a Map.

Attribute	Description	Values
mapped-by	When the map is formed by a foreign-key, the value can be a field in a key persistable class. This attribute defines which field in the key class is used as the value.	
column	Name of the column (if only one)	
delete-action	Action to be performed when the owner object is deleted. This is to be specified when defining foreign key information	cascade, restrict, null, default, none
indexed	Whether the value column should be indexed. This is to be specified when defining index information	true, false, unique
unique	Whether the value column should be unique. This is to be specified when defining unique key information	true, false
converter	Class name of a converter class (AttributeConverter) to use for this key.	
use-default-conversion	Whether we should just use any default conversion (defined via persistent properties)	true, **false**

Metadata for order tag

These are attributes within the <**order**> tag (jdo/package/class/field/order). This is used to define the column details for the ordering column in a List.

Attribute	Description	Values
mapped-by	When a List is formed by a foreign-key, the ordering can be a field in the element persistable class. This attribute defines which field in the element class is used as the ordering. The field must be of type *int*, *Integer*, *long*, *Long*. DataNucleus will write the index positions to this field (starting at 0 for the first item in the List)	
column	Name of the column to use for ordering.	

These are attributes within the <**extension**> tag (jdo/package/class/field/order/extension).

Attribute	Description	Values
list-ordering	Used to make the list be an "ordered list" where it has no index column and instead will order the elements by the specified expression upon retrieval. The ordering expression takes names and ASC/DESC and can be a composite	{orderfield [ASC

Metadata for index tag

These are attributes within the <**index**> tag (jdo/package/class/field/index). This element is used where a user wishes to add specific indexes to the datastore to provide more efficient access to particular fields.

Attribute	Description	Values
name	Name of the index in the datastore	
unique	Whether the index is unique	true, false
column	Name of the column to use (alternative to specifying it as a sub-element).	

These are attributes within the <**extension**> tag (jdo/package/class/field/index/extension).

Attribute	Description	Values
extended-setting	Additional settings appended to the end of the CREATE INDEX statement (depends on the precise syntax allowed by the RDBMS).	

Metadata for foreign-key tag

These are attributes within the <**foreign-key**> tag (jdo/package/class/field/foreign-key). This is used where the user wishes to define the behaviour of the foreign keys added due to the relationships in the object model. This is to be read in conjunction with foreign-key guide

Attribute	Description	Values
name	Name of the foreign key in the datastore	
deferred	Whether the constraints are initially deferred.	true, false
delete-action	Action to be performed when the owner object is deleted.	cascade, restrict, null, default
update-action	Action to be performed when the owner object is updated.	cascade, restrict, null, default

Metadata for unique tag

These are attributes within the <**unique**> tag (jdo/package/class/unique, jdo/package/class/field/unique). This element is used where a user wishes to add specific unique constraints to the datastore to provide more control over particular fields.

Attribute	Description	Values
name	Name of the constraint in the datastore	
column	Name of the column to use (alternative to specifying it as a sub-element).	

Metadata for column tag

These are attributes within the <**column**> tag (*/column). This is used to define the details of a column in the datastore, and so can be used to match to an existing datastore schema.

Attribute	Description	Values
name	Name of the column in the datastore. See also the property **datanucleus.identifier.case** in the PMF Properties Guide.	
length	Length of the column in the datastore (for character types), or the precision of the column in the datastore (for floating point field types).	positive integer
scale	Scale of the column in the datastore (for floating point field types).	positive integer
jdbc-type	JDBC Type to use for this column in the datastore when the default value is not satisfactory. Please refer to JDBC for the valid types. Not all of these types are supported for all RDBMS mappings.	Valid JDBC Type (CHAR, VARCHAR, LONGVARCHAR, NUMERIC, DECIMAL, BIT, TINYINT, SMALLINT, INTEGER, BIGINT, REAL, FLOAT, DOUBLE, BINARY, VARBINARY, LONGVARBINARY, DATE, TIME, TIMESTAMP, BLOB, BOOLEAN, CLOB, DATALINK)
sql-type	SQL Type to use for this column in the datastore. This should not usually be necessary since the specification of JDBC type together with length/scale will likely define it.	Valid SQL Type (e.g VARCHAR, CHAR, NUMERIC etc)
allows-null	Whether the column in the datastore table should allow nulls or not. The default is "false" for primitives, and "true" otherwise.	true, false
default-value	Default value to use for this column when creating the table. If you want the default to be NULL, then put this as "#NULL". This is particularly for cases where you have a table that stores multiple classes in an inheritance tree (subclass-table, superclass-table) so when you persist a superclass object it doesn't have the subclass fields in its INSERT and so the datastore uses the default-value settings that are embodied in the CREATE TABLE statement.	Default value expression

Attribute	Description	Values
target	Declares the name of the primary key column for the referenced table. For columns contained in join elements, this is the name of the primary key column in the primary table. For columns contained in field, element, key, value, or array elements, this is the name of the primary key column of the primary table of the other side of the relationship.	target column name
target-field	Declares the name of the primary key field for the referenced class. For columns contained in join elements, this is the name of the primary key field in the base class. For columns contained in field, element, key, value, or array elements, this is the name of the primary key field of the base class of the other side of the relationship.	target field name
insert-value	Value to use for this column when it has no field in the class and an object is being inserted. If you want the inserted value to be NULL, then put this as "#NULL"	Insert value
position	Position of the column in the table (0 = first).	positive integer

These are attributes within the <**extension**> tag (*/column/extension).

Attribute	Description	Values
column-mapping-class	Specifies the (RDBMS) column mapping class to be used for mapping this field. This is only used where the user wants to override the default DataNucleus column mapping class and provide their own mapping class for this field based on the database data type. This column mapping class must be available in the DataNucleus PersistenceManagerFactory CLASSPATH.	Fully-qualified class name
enum-check-constraint	Specifies that a CHECK constraint for this column must be generated based on the values of a java.lang.Enum type. e.g. enum Color (RED, GREEN, BLUE) where its name is persisted a CHECK constraint is defined as *CHECK "COLUMN" IN ('RED', 'GREEN', 'BLUE')*.	true, false

Metadata for join tag

These are attributes within the <**join**> tag (jdo/package/class/field/join). This element is added when the field has a mapping to a "join" table (as part of a 1-N relationship). It is also used to specify overriding of details in an inheritance tree where the primary key columns are shared up the hierarchy. A further use (when specified under the <class> element) is for specifying the column details for joining to a Secondary Table.

Attribute	Description	Values
column	Name of the column used to join to the PK of the primary table (when only one column used). Used in Secondary Tables.	
table	Table name used when joining the PK of a FCO class table to a secondary table. See Secondary Tables.	
delete-action	Action to be performed when the owner object is deleted. This is to be specified when defining foreign key information	cascade, restrict, null, default, none
indexed	Whether the join table owner column should be indexed. This is to be specified when defining index information	true, false, unique
unique	Whether the join table owner column should be unique. This is to be specified when defining unique key information	true, false
outer	Whether to use an outer join here. This is of particular relevance to secondary tables	true, **false**

These are attributes within the <**extension**> tag (jdo/package/class/field/join/extension). These are for controlling the join table.

Attribute	Description	Values
primary-key	This parameter defines if the join table will be assigned a primary key. The default is true since it is considered a best practice to have primary keys on all tables. This allows the option of turning it off.	**true**, false

Metadata for element tag

These are attributes within the <**element**> tag (jdo/package/class/field/element). This element is added when the field has a mapping to a "element" (as part of a 1-N relationship).

Attribute	Description	Values
mapped-by	The name of the field at the other ("N") end of a relationship when this field is the "1" side of a 1-N relationship (for FK relationships). This performs the same function as specifying "mapped-by" on the <field> element.	
column	Name of the column (alternative to specifying it as a sub-element).	
delete-action	Action to be performed when the owner object is deleted. This is to be specified when defining foreign key information	cascade, restrict, null, default, none
indexed	Whether the element column should be indexed. This is to be specified when defining index information	true, false, unique
unique	Whether the element column should be unique. This is to be specified when defining unique key information	true, false
converter	Class name of a converter class (AttributeConverter) to use for this key.	
use-default-conversion	Whether we should just use any default conversion (defined via persistent properties)	true, **false**

Metadata for collection tag

These are attributes within the <**collection**> tag (jdo/package/class/field/collection). This is used to define the persistence of a Collection.

Attribute	Description	Values
element-type	The type of element stored in this Collection or array (fully qualified class). This is not required when the field is an array. It is also not required when the Collection is defined using Java generics.	
embedded-element	Whether the elements of a collection or array-valued persistent field should be stored embedded or as first-class objects. It's a hint for the JDO implementation to store, if possible, the elements of the collection as part of the it instead of as their own instances in the datastore. See the <embedded> element for details on how to define the field mappings for the embedded element.	true, false
dependent-element	Whether the elements of the collection are to be considered dependent on the owner object.	true, **false**
serialized-element	Whether the elements of a collection or array-valued persistent field should be stored serialised into a single column of the join table (where used).	true, **false**

These are attributes within the <**extension**> tag (jdo/package/class/field/collection/extension).

Attribute	Description	Values
cache	Whether this SCO collection will be cached by DataNucleus or whether every access of the collection will go through to the datastore. See also the property **datanucleus.cache.collections** in the PMF Properties Guide. This MetaData attribute is used to override the value used by the *PersistenceManagerFactory*	**true**, false
cache-lazy-loading	Whether objects from this SCO collection will be lazy loaded (loaded when required) or whether they should be loaded at initialisation. See also the property **datanucleus.cache.collections.lazy** in the PMF Properties Guide. This MetaData attribute is used to override the value used by the *PersistenceManagerFactory*	**true**, false
comparator-name	Defines the name of the comparator to use with SortedSet, TreeSet collections. The specified name is the name of the comparator class, which must have a default constructor. This extension is only used by SortedSet, TreeSet fields.	Fully-qualified class name

Metadata for map tag

These are attributes within the <**map**> tag (jdo/package/class/field/map). This is used to define the persistence of a Map.

Attribute	Description	Values
key-type	The type of key stored in this Map (fully qualified class). This is not required when the Map is defined using Java generics.	
embedded-key	Whether the elements of a Map key field should be stored embedded or as first-class objects.	true, false
value-type	The type of value stored in this Map (fully qualified class). This is not required when the Map is defined using Java generics.	
embedded-value	Whether the elements of a Map value field should be stored embedded or as first-class objects.	true, false
dependent-key	Whether the keys of the map are to be considered dependent on the owner object.	true, **false**
dependent-value	Whether the value of the map are to be considered dependent on the owner object.	true, **false**
serialized-key	Whether the keys of a map-valued persistent field should be stored serialised into a single column of the join table (where used).	true, **false**
serialized-value	Whether the values of a map-valued persistent field should be stored serialised into a single column of the join table (where used).	true, **false**

These are attributes within the <**extension**> tag (jdo/package/class/field/map/extension).

Attribute	Description	Values
cache	Whether this SCO map will be cached by DataNucleus or whether every access of the map will go through to the datastore. See also "datanucleus.cache.collections" in the PMF Properties Guide. This MetaData attribute is used to override the value used by the *PersistenceManagerFactory*	**true**, false
cache-lazy-loading	Whether objects from this SCO map will be lazy loaded (loaded when required) or whether they should be loaded at initialisation. See also "datanucleus.cache.collections.lazy" in the PMF Properties Guide. This MetaData attribute is used to override the value used by the *PersistenceManagerFactory*	**true**, false

Attribute	Description	Values
comparator-name	Defines the name of the comparator to use with SortedMap, TreeMap maps. The specified name is the name of the comparator class, which must have a default constructor. This extension is only used by SortedMap, TreeMap fields.	Fully-qualified class name

Metadata for array tag

This is used to define the persistence of an array. DataNucleus provides support for many types of arrays, either serialised into a single column, using a join table, or via a foreign-key (for arrays of PC objects).

Attribute	Description	Values
embedded-element	Whether the array elements should be stored embedded (default = true for primitives, wrappers etc and false for persistable objects).	true, false
serialized-element	Whether the array elements should be stored serialised into a single column in the join table.	true, **false**
dependent-element	Whether the elements of the array are to be considered dependent on the owner object.	true, **false**

Metadata for sequence tag

These are attributes within the <**sequence**> tag. This is used to denote a JDO datastore sequence.

Attribute	Description	Values
name	Symbolic name for the sequence for this package	
datastore-sequence	Name of the sequence in the datastore	
factory-class	Factory class for creating the sequence. Please refer to the Sequence guide	
strategy	Strategy to use for application of this sequence.	nontransactional, contiguous, noncontiguous
allocation-size	Allocation size for the sequence for this package	50
initial-value	Initial value for the sequence for this package	1

These are attributes within the <**extension**> tag (jdo/package/class/sequence/extension). These are for controlling the datastore sequences created by DataNucleus. Please refer to the documentation for the value generator being used for applicability

Attribute	Description	Values
sequence-catalog-name	The catalog used to store sequences for use by value generators. See Value Generation. Default catalog for the datastore will be used if not specified.	
sequence-schema-name	The schema used to store sequences for use by value generators. See Value Generation. Default schema for the datastore will be used if not specified.	
sequence-table-name	The table used to store sequences for use by value generators. See Value Generation.	**SEQUENCE_TABLE**
sequence-name-column-name	The column name in the sequence-table used to store the name of the sequence for use by value generators. See Value Generation.	**SEQUENCE_NAME**
sequence-nextval-column-name	The column name in the sequence-table used to store the next value in the sequence for use by value generators. See Value Generation.	**NEXT_VAL**
key-min-value	The minimum key value for use by value generators. Keys lower than this will not be generated. See Value Generation.	
key-max-value	The maximum key value for use by value generators. Keys higher than this will not be generated. See Value Generation.	

Attribute	Description	Values
key-initial-value	The starting value for use by value generators. Keys will start from this value when being generated. See Value Generation.	
key-cache-size	The cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	
key-database-cache-size	The database cache size for keys for use by value generators. The cache of keys will be constrained by this value. See Value Generation.	

Metadata for fetch-plan tag

These are attributes within the <**fetch-plan**> tag (jdo/fetch-plan). This element is used to define fetch plans that are utilised at runtime, and are of particular use with queries. This element contains **fetch-group** sub-elements.

Attribute	Description	Values
name	Name of the fetch plan.	
maxFetchDepth	Max depth to fetch with this fetch plan	1
fetchSize	Size to fetch with this fetch plan (for use with query result sets	0

Metadata for class extension tag

These are attributes within the <**extension**> tag (jdo/package/class/extension). These are for controlling the class definition

Attribute	Description	Values
requires-table	This is for use with a "nondurable" identity case and specifies whether the class requires a table/view in the datastore.	**true**, false
ddl-definition	Definition of the TABLE SCHEMA to be used by the class.	**true**, false
ddl-imports	Classes imported resolve macro identifiers in the definition of a RDBMS Table.	
mysql-engine-type	"Engine Type" to use when creating the table for this class in MySQL. Refer to the MySQL documentation for ENGINE type (e.g INNODB, MEMORY, ISAM)	
view-definition	Definition of the VIEW to be used by the class. Please refer to the RDBMS Views Guide for details. If your view already exists, then specify this as " " and have the autoStart flags set to false.	
view-imports	Classes imported resolve macro identifiers in the definition of a RDBMS View. Please refer to the RDBMS Views Guide for details.	
read-only	Whether objects of this type are read-only. Setting this to true will prevent any insert/update/delete of this type	true, **false**

Metadata for extension tag

These are attributes within the <**extension**> tag. This is used to denote a DataNucleus extension to JDO.

Attribute	Description	Values
vendor-name	Name of the vendor. For DataNucleus we use the name "datanucleus" (lowercase).	
key	Key of the extension property	
value	Value of the extension property	